

File - IteratorObserver.java

```
1 package ch.heig.sio.lab2.groupD.Utilities;
2
3 import ch.heig.sio.lab2.tsp.Edge;
4
5 import java.util.Iterator;
6 import java.util.NoSuchElementException;
7
8 /**
9  * Iterateur utilisé pour parcourir un tableau de villes et créer des arêtes à la volée
10 * On l'utilise pour que l'observateur puisse créer les arêtes seulement si besoin
11 *
12 * @author Edwin Häffner
13 * @author Arthur Junod
14 */
15 public class IteratorObserver implements Iterator<Edge> {
16
17     public IteratorObserver(int[] tour){
18         this.tour = tour;
19         this.length = tour.length;
20     }
21
22     @Override
23     public boolean hasNext() {
24         return index < length;
25     }
26
27     @Override
28     public Edge next() {
29         if(hasNext()){
30             return new Edge(tour[index], tour[++index % length]);
31         } else {
32             throw new NoSuchElementException();
33         }
34     }
35
36     private final int[] tour;
37     private final int length;
38     private int index = 0;
39 }
```

File - Improvement20pt.java

```
1 package ch.heig.sio.lab2.groupD;
2
3 import ch.heig.sio.lab2.display.ObservableTspImprovementHeuristic;
4 import ch.heig.sio.lab2.display.TspHeuristicObserver;
5 import ch.heig.sio.lab2.groupD.Utilities.IteratorObserver;
6 import ch.heig.sio.lab2.tsp.TspData;
7 import ch.heig.sio.lab2.tsp.TspTour;
8
9 /**
10 * <p>
11 * Cette classe implémente l'heuristique d'amélioration 2-opt pour le problème du voyageur de commerce.
12 * Pour trouver cette amélioration, la fonction computeTour() parcourt toutes les combinaisons possibles de deux arêtes
13 * dans le tour initial et les échange si cela permet de réduire la longueur du tour. On utilise la stratégie du "best fit".
14 * C'est-à-dire qu'on force l'algorithme à parcourir toutes les arêtes du tour pour trouver la meilleure amélioration possible.
15 * </p>
16 * <p>
17 * computeTour() continue de tourner tant qu'une amélioration est trouvée. Si aucune amélioration n'est trouvée, l'algorithme s'arrête et retourne le tour amélioré.
18 * </p>
19 * <p>
20 * La complexité de cet algorithme est de  $O(q \cdot n^2)$  où  $q$  est le nombre d'améliorations trouvées et  $n$  le nombre de sommets.
21 * On ne peut pas savoir à l'avance combien d'améliorations seront trouvées.
22 * </p>
23 *
24 *
25 * @author Edwin Häffner
26 * @author Arthur Junod
27 */
28 public class Improvement20pt implements ObservableTspImprovementHeuristic {
29
30     /**
31      * Cette fonction permet de 2-optimiser un tour donné.
32      * Elle tourne en boucle jusqu'à ne plus pouvoir améliorer ce tour et va toujours prendre la meilleure 2-optimisation
33      * possible à chaque itération.
34      *
35      * @param initialTour    Le tour à améliorer
36      * @param observer        L'Observer utilisé pour mettre à jour visuellement l'application
37      * @return                 Un nouveau tour qui est 2-optimisé
38      */
39     @Override
40     public TspTour computeTour(TspTour initialTour, TspHeuristicObserver observer) {
41
42         //Copie des données du tour initial
43         final TspData tourData = initialTour.data();
44         final int tourNbVertices = initialTour.tour().size();
45         final int[] tourCopy = initialTour.tour().copy();
46
47         long tourLength = initialTour.length();
48         long oldTourLength; Utilisez un booléen (ex. "improved"), même idée mais plus explicite.
49
50         //Boucle principale de l'algorithme améliorant. Il s'arrête lorsque plus aucune amélioration n'est trouvée.
51         do {
52             oldTourLength = tourLength; //Sauvegarde de la longueur du tour avant amélioration
53
54             //Mise des variables de la meilleure amélioration à -1 pour éviter un warning
55             int bestI = -1;
56             int bestJ = -1;
57             long bestImprovement = 0;
58
59         } while (bestImprovement > 0);
60
61         return new TspTour(tourCopy);
62     }
63 }
```

File - Improvement20pt.java

```
59         // On s'arrête deux sommets avant la fin, car les arcs qui suivent ont déjà été
60         traité.
61         for (int i = 0; i < tourNbVertices - 2; i++) {
62             int i1 = i + 1;
63             int currentEdge1 = tourData.getDistance(tourCopy[i], tourCopy[i1]);
64
65             for (int j = i + 2; j < tourNbVertices; j++) {
66                 int j1 = j + 1;
67                 // Dans le cas où j1 atteint la fin du tour, on le fait pointer vers le
68                 premier sommet
69                 // Comme j est toujours < tourNbVertices, j1 ne peut valoir que
70                 tourNbVertices
71                 // Cette approche est plus performante qu'utiliser un modulo
72                 if (j1 == tourNbVertices) {
73                     j1 = 0;
74                 }
75
76                 //Calcul de l'amélioration possible
77                 long currentCost = currentEdge1 + tourData.getDistance(tourCopy[j], tourCopy[
78                     j1]);
79
80                 long newCost = tourData.getDistance(tourCopy[i], tourCopy[j]) +
81                     tourData.getDistance(tourCopy[i1], tourCopy[j1]);
82                 long improvement = currentCost - newCost;
83
84                 //Si une amélioration est trouvée, on garde sa valeur et on indique quels
85                 sommets à inverser
86                 if (improvement > bestImprovement) {
87                     bestImprovement = improvement;
88                     bestI = i;
89                     bestJ = j;
90                 }
91             }
92
93             if (bestImprovement > 0) {
94                 // Inversion des sommets entre i+1 et j en utilisant une approche "deux pointeurs"
95
96                 // qui se déplacent l'un vers l'autre jusqu'à se rencontrer
97                 int i = bestI + 1;
98                 int j = bestJ;
99                 int temp;
100                //Swap des sommets entre i et j (i et j inclus)
101                for (;i < j; ++i){
102                    temp = tourCopy[i];
103                    tourCopy[i] = tourCopy[j];
104                    tourCopy[j] = temp;
105                    --j;
106                }
107                tourLength -= bestImprovement; //Mise à jour de la longueur du tour
108                observer.update(new IteratorObserver(tourCopy));
109            }
110        } while (oldTourLength != tourLength); //True tant qu'on trouve une nouvelle amélioration
111
112        return new TspTour(tourData, tourCopy, tourLength);
113    }
114 }
```

```

1 package ch.heig.sio.lab2.groupD;
2
3
4 import ch.heig.sio.lab2.groupD.heuristics.ClosestFirstInsert;
5 import ch.heig.sio.lab2.groupD.heuristics.FarthestFirstInsert;
6 import ch.heig.sio.lab2.tsp.RandomTour;
7 import ch.heig.sio.lab2.tsp.TspConstructiveHeuristic;
8 import ch.heig.sio.lab2.tsp.TspData;
9 //import io.github.cdimascio.dotenv.Dotenv;
10
11 import java.text.DecimalFormat;
12 import java.util.*;
13
14 import static java.util.Collections.max;
15 import static java.util.Collections.min;
16
17 /**
18 * <p>
19 * Cette classe permet de faire l'analyse des heuristiques sans affichage gui.
20 * On récupère des statistiques sur les distances calculées telles que la médiane, la moyenne, la
21 * dérivation standard,
22 * la distance maximale et la distance minimale pour chaque heuristique avant et après la 2-
23 * optimisation et par fichier de données.
24 * Finalement, on affiche ces statistiques avec un calcul de la performance pour chaque
25 * indicateur (qui est la distance de l'indicateur comparée à la distance optimale, plus elle est
26 * proche de 100% mieux c'est)
27 * ainsi que le temps d'exécution de l'heuristique avec la 2-optimisation.
28 * </p>
29 * <p>
30 * Exemple d'affichage :
31 * </p>
32 * <pre>
33 * Analysis for dataset: pcb442
34 * Optimal tour length: 50'778
35 * Metric           ClosestFirstInsert      FarthestFirstInsert      RandomTour
36 * -----
```

Metric	ClosestFirstInsert	FarthestFirstInsert	RandomTour
Min	59'368.00 (116.92%)	56'016.00 (110.32%)	727'438.00 (1'432.58%)
Median	60'362.50 (118.88%)	57'266.50 (112.78%)	770'826.00 (1'518.03%)
Mean	60'385.48 (118.92%)	57'416.56 (113.07%)	773'247.40 (1'522.80%)
Max	61'351.00 (120.82%)	59'875.00 (117.92%)	797'408.00 (1'570.38%)
StdDev	510.08	797.87	15'306.18
Min2opt	54'373.00 (107.08%)	54'914.00 (108.15%)	53'916.00 (106.18%)
Median2opt	55'439.50 (109.18%)	56'671.50 (111.61%)	56'326.50 (110.93%)
Mean2opt	55'294.62 (108.89%)	56'727.72 (111.72%)	56'286.34 (110.85%)
Max2opt	55'969.00 (110.22%)	59'275.00 (116.73%)	58'183.00 (114.58%)
StdDev2opt	394.39	882.80	880.88
MeanTime (ms)	32.08	9.44	135.20

37 * </pre>
38 * <p>
39 * The percentage next to the values are an indication of the performance of the heuristic.
40 * The performance is a percentage of the mean distance compared to the optimal distance.
41 * It should never be below 100% and the closer to 100% the better.
42 * </p>
43 * @author Edwin Häffner
44 * @author Arthur Junod
45 */
46
47 public final class Analyze {
48 // Record pour les statistiques que l'on va afficher.
49 private record Statistics(double min, double median, double mean, double max, double stdDev,
50 double min2opt, double median2opt, double mean2opt, double max2opt,
51 double stdDev2opt,
52 double meanTime) {}
53
54 // Variables statiques pour la graine aléatoire du RandomTour et du nombre de villes (et donc d'
55 // optimisation) que l'on va faire.
56 static long RANDOM_SEED;

File - Analyze.java

```

62 static final int NUMBER_CITIES = 50;
63
64
65 public static void main(String[] args) {
66     // Longueurs optimales :
67     // pcb442 : 50778
68     // att532 : 86729
69     // u574 : 36905
70     // pcb1173 : 56892
71     // nrw1379 : 56638
72     // u1817 : 57201
73
74     // Chargement des variables d'environnement pour éviter de devoir recompiler la classe lors
    de changement de seed.
75     // A besoin de la dépendance dotenv-java dans Maven
76     try {
77         Dotenv dotenv = Dotenv.configure().load();
78         RANDOM_SEED = Long.parseLong(dotenv.get("TSP_SEED"), 16);
79     } catch (Exception e) {
80         System.err.println("Error loading environment variables, make sure you have a .env file
        with TSP_SEED set. Using default value of 0x134DAE9.");
81         RANDOM_SEED = 0x134DAE9;
82     }
83     RANDOM_SEED = 0x134DAE9;
84
85     TspConstructiveHeuristic[] heuristics = {
86         new ClosestFirstInsert(),
87         new FarthestFirstInsert(),
88         new RandomTour(RANDOM_SEED) Nouvelle instance à créer pour chaque jeu de données.
89     };
89
90
91     var opt2 = new Improvement20pt();
92
93     // Tableau des fichiers de données
94     String[] files = {"pcb442", "att532", "u574", "pcb1173", "nrw1379", "u1817"};
95     long[] optimalDistances = {50778, 86729, 36905, 56892, 56638, 57201};
96
97     System.out.println("Analyzing heuristics...");
98
99     // Boucle sur les fichiers
100    for (int fileIndex = 0; fileIndex < files.length; fileIndex++) {
101        String file = files[fileIndex];
102
103        // Ouvre chaque fichier et fait une analyse des heuristiques
104        try {
105            TspData data = TspData.fromFile("data/" + file + ".dat");
106
107            System.out.println("\nProcessing dataset: " + file + ".dat (" + data.getNumberOfCities
() + " cities)");
108            Map<String, Statistics> stats = new LinkedHashMap<>();
109
110            // Ce RandomTour est utilisé pour générer les NUMBER_CITIES villes de départ pour les
    heuristiques qui en ont besoin.
111            var randomTour = new RandomTour(RANDOM_SEED);
112            var cities = randomTour.computeTour(data, 0).tour();
113
114            // Boucle sur les heuristiques
115            for (var heuristic : heuristics) {
116                ArrayList<Long> results = new ArrayList<>();
117                ArrayList<Long> resultsWithImprovement = new ArrayList<>();
118
119                long meanValue = 0;
120                long meanValueWithImprovement = 0;
121                long meanValueTime = 0;
122                // Boucle sur les NUMBER_CITIES villes
123                for (int i = 0; i < NUMBER_CITIES; ++i) {

```

```

124         var timeBefore = System.currentTimeMillis(); System.nanoTime() est la méthode idomatiche en Java pour calculer des écarts de temps (mais ok).
125         // Compute le tour initial avec l'heuristique
126         var tourHeuristic = heuristic.computeTour(data, cities.get(i));
127         // Compute le tour amélioré grâce 2-opt
128         var tour20pt = opt2.computeTour(tourHeuristic);
129         var timeExec = System.currentTimeMillis() - timeBefore;
130         // Extraction des longueurs des deux tours
131         long length20pt = tour20pt.length();
132         long lengthHeuristic = tourHeuristic.length();
133
134         results.add(lengthHeuristic);
135         resultsWithImprovement.add(length20pt);
136
137         meanValue += lengthHeuristic;
138         meanValueWithImprovement += length20pt;
139         meanValueTime += timeExec;
140
141         // Mise à jour de la barre de progression affichée
142         updateProgress(i + 1, NUMBER_CITIES, heuristic.getClass().getSimpleName());
143     }
144
145
146     // Calcul des statistiques
147     double mean = (double) meanValue / NUMBER_CITIES;
148     double meanImprovement = (double) meanValueWithImprovement / NUMBER_CITIES;
149     double medianValue = median(results);
150     double medianValueImprovement = median(resultsWithImprovement);
151     double stdDevValue = stdDev(results, mean);
152     double stdDevValueImprovement = stdDev(resultsWithImprovement, meanImprovement);
153     double meanTime = (double) meanValueTime / NUMBER_CITIES;
154
155     stats.put(heuristic.getClass().getSimpleName(), new Statistics(
156             min(results),
157             medianValue,
158             mean,
159             max(results),
160             stdDevValue,
161             min(resultsWithImprovement),
162             medianValueImprovement,
163             meanImprovement,
164             max(resultsWithImprovement),
165             stdDevValueImprovement,
166             meanTime
167         ));
168     }
169
170     printStatistics(file, stats, optimalDistances[fileIndex]);
171
172 } catch (Exception e) {
173     System.err.println("There was an error in processing " + file + ".dat");
174     System.err.println(e.getMessage());
175     return;
176 }
177 }
178 }
179
180 /**
181 * Calcule la médiane d'une liste de valeurs.
182 *
183 * @param values La liste de valeurs
184 * @return La médiane des valeurs
185 */
186 public static double median(List<Long> values) {
187     Collections.sort(values);
188     int middle = values.size() / 2;
189     if (values.size() % 2 == 0) {

```

File - Analyze.java

```

190     return (values.get(middle - 1) + values.get(middle)) / 2.0;
191 } else {
192     return values.get(middle);
193 }
194 }
195
196 /**
197 * Calcule l'écart-type d'une liste de valeurs.
198 *
199 * @param values La liste de valeurs
200 * @param mean La moyenne des valeurs
201 * @return L'écart-type des valeurs
202 */
203 public static double stdDev(List<Long> values, double mean) {
204     double sum = 0;
205     for (Long value : values) {
206         sum += Math.pow(value - mean, 2);
207     }
208     return Math.sqrt(sum / (values.size() - 1));
209 }
210
211 /**
212 * Affiche les statistiques pour les heuristiques en format lisible.
213 * Généré par ClaudeAI
214 *
215 * @param filename Le nom du fichier analysé
216 * @param heuristicStats Les statistiques pour chaque heuristique
217 * @param optimalDistance La meilleure distance possible pour le dataset
218 */
219 private static void printStatistics(String filename, Map<String, Statistics> heuristicStats,
220 long optimalDistance) {
221     int metricWidth = 20;
222     int valueWidth = 35;
223
224     // Affichage du header des statistiques
225     System.out.println("\nAnalysis for dataset: " + filename);
226     System.out.println("Optimal tour length: " + String.format("%,d", optimalDistance));
227     System.out.printf("%-" + metricWidth + "s", "Metric");
228     for (String heuristic : heuristicStats.keySet()) {
229         System.out.printf("%-" + valueWidth + "s", heuristic);
230     }
231     System.out.println();
232
233     // Séparateur
234     System.out.println("-".repeat(metricWidth + (valueWidth * heuristicStats.size())));
235
236     String[] metrics = {"Min", "Median", "Mean", "Max", "StdDev",
237                         "Min2opt", "Median2opt", "Mean2opt", "Max2opt", "StdDev2opt",
238                         "MeanTime (ms)"};
239
240     DecimalFormat df = new DecimalFormat("#,##0.00");
241     DecimalFormat pctFormat = new DecimalFormat("#,##0.00%");
242
243     for (String metric : metrics) {
244         System.out.printf("%-" + metricWidth + "s", metric);
245         for (Statistics stats : heuristicStats.values()) {
246             double value = switch (metric) {
247                 case "Min" -> stats.min;
248                 case "Median" -> stats.median;
249                 case "Mean" -> stats.mean;
250                 case "Max" -> stats.max;
251                 case "StdDev" -> stats.stdDev;
252                 case "Min2opt" -> stats.min2opt;
253                 case "Median2opt" -> stats.median2opt;
254                 case "Mean2opt" -> stats.mean2opt;
255                 case "Max2opt" -> stats.max2opt;
256             }
257             System.out.printf("%-" + valueWidth + "s", value);
258         }
259         System.out.println();
260     }
261 }

```

File - Analyze.java

```

255         case "StdDev2opt" -> stats.stdDev2opt;
256         case "MeanTime (ms)" -> stats.meanTime;
257         default -> 0.0;
258     };
259
260     // Ne pas afficher le pourcentage pour les métriques de temps et d'écart type
261     if (metric.equals("MeanTime (ms)") || metric.contains("StdDev")) {
262         System.out.printf("%-" + valueWidth + "s", df.format(value));
263     } else {
264         double percentage = value / optimalDistance;
265         System.out.printf("%-" + valueWidth + "s",
266                           df.format(value) + " (" + pctFormat.format(percentage) + ")");
267     }
268 }
269 System.out.println();
270 }
271 System.out.println("\nThe percentage next to the values are an indication of the performance
of the heuristic.");
272 System.out.println("The performance is a percentage of the distance compared to the optimal
distance.\nIt should never be below 100% and the closer to 100% the better.");
273 }
274
275 /**
276 * Affiche une barre de progression pour suivre l'avancement de l'analyse.
277 * Généré par ClaudeAI
278 * @param current           La ville actuellement traitée
279 * @param total              Le nombre total de villes à traiter
280 * @param currentHeuristic  Le nom de l'heuristique actuellement utilisée
281 */
282 private static void updateProgress(int current, int total, String currentHeuristic) {
283     int progressBarWidth = 40;
284     double percentage = (double) current / total * 100;
285     int completedWidth = progressBarWidth * current / total;
286
287     // Crée la barre de progression
288     StringBuilder progressBar = new StringBuilder("[");
289     for (int i = 0; i < progressBarWidth; i++) {
290         if (i < completedWidth) {
291             progressBar.append "=";
292         } else if (i == completedWidth) {
293             progressBar.append ">";
294         } else {
295             progressBar.append " ";
296         }
297     }
298     progressBar.append "]";
299
300     // Affiche la barre de progression et le pourcentage
301     System.out.print("\r" + currentHeuristic + " Progress: " + progressBar + " " +
302                      String.format("%.1f%%", percentage) + "    "); // Ajout de plusieurs espaces pour
effacer les anciennes valeurs
303
304     // Ajout d'un retour à la ligne si on a fini
305     if (current == total) {
306         System.out.println();
307     }
308 }
309 }
310
311

```

```

1 Analyzing heuristics...
2
3 Analysis for dataset: pcb442
4 Optimal tour length: 50'778
5 Metric          ClosestFirstInsert          FarthestFirstInsert
6 RandomTour
6 -----
7 Min           59'368.00 (116.92%)        56'016.00 (110.32%)        727'438.
8 Median         60'362.50 (118.88%)        57'266.50 (112.78%)        770'826.
9 Mean           60'385.48 (118.92%)        57'416.56 (113.07%)        773'247.
10 Max            61'351.00 (120.82%)        59'875.00 (117.92%)        797'408.
11 StdDev        510.08
12 StdDev        510.08
12 Min2opt       54'373.00 (107.08%)        54'914.00 (108.15%)        53'916.
13 Median2opt    55'439.50 (109.18%)        56'671.50 (111.61%)        56'326.
14 Mean2opt      55'294.62 (108.89%)        56'727.72 (111.72%)        56'286.
15 Max2opt       55'969.00 (110.22%)        59'275.00 (116.73%)        58'183.
16 StdDev2opt    394.39
17 MeanTime (ms) 20.38
18
19 The percentage next to the values are an indication of the performance of the heuristic.
20 The performance is a percentage of the distance compared to the optimal distance.
21 It should never be below 100% and the closer to 100% the better.
22
23 Analysis for dataset: att532
24 Optimal tour length: 86'729
25 Metric          ClosestFirstInsert          FarthestFirstInsert
26 RandomTour
26 -----
27 Min           106'739.00 (123.07%)        92'951.00 (107.17%)        1'561'
28 Median         107'501.50 (123.95%)        94'860.00 (109.38%)        1'615'
29 Mean           107'594.38 (124.06%)        94'992.80 (109.53%)        1'618'
30 Max            108'738.00 (125.38%)        97'545.00 (112.47%)        1'686'
31 StdDev        488.13
32 StdDev        488.13
32 Min2opt       96'219.00 (110.94%)        92'236.00 (106.35%)        92'118.
33 Median2opt    97'360.50 (112.26%)        93'721.00 (108.06%)        94'491.
34 Mean2opt      97'252.58 (112.13%)        93'849.00 (108.21%)        94'538.
35 Max2opt       98'476.00 (113.54%)        96'582.00 (111.36%)        96'879.
36 StdDev2opt    655.61
37 MeanTime (ms) 40.80
38
39 The percentage next to the values are an indication of the performance of the heuristic.
40 The performance is a percentage of the distance compared to the optimal distance.
41 It should never be below 100% and the closer to 100% the better.
42
43 Analysis for dataset: u574
44 Optimal tour length: 36'905

```

45 Metric	ClosestFirstInsert	FarthestFirstInsert	
46 RandomTour			
47 Min .00 (1'782.17%)	45'013.00 (121.97%)	39'773.00 (107.77%)	657'710
48 Median .00 (1'854.02%)	45'509.50 (123.32%)	40'767.00 (110.46%)	684'227
49 Mean .62 (1'856.93%)	45'511.06 (123.32%)	40'786.84 (110.52%)	685'301
50 Max .00 (1'918.56%)	46'071.00 (124.84%)	42'027.00 (113.88%)	708'046
51 StdDev 04	315.69	438.84	11'018.
52 Min2opt 00 (107.18%)	40'277.00 (109.14%)	39'253.00 (106.36%)	39'553.
53 Median2opt 00 (109.55%)	40'718.50 (110.33%)	40'296.50 (109.19%)	40'431.
54 Mean2opt 56 (109.70%)	40'705.26 (110.30%)	40'264.20 (109.10%)	40'483.
55 Max2opt 00 (112.90%)	41'078.00 (111.31%)	41'247.00 (111.77%)	41'666.
56 StdDev2opt	167.39	433.25	434.73
57 MeanTime (ms)	48.04	12.60	287.80
58			
59 The percentage next to the values are an indication of the performance of the heuristic.			
60 The performance is a percentage of the distance compared to the optimal distance.			
61 It should never be below 100% and the closer to 100% the better.			
62			
63 Analysis for dataset: pcb1173			
64 Optimal tour length: 56'892			
65 Metric	ClosestFirstInsert	FarthestFirstInsert	
66 RandomTour			
67 Min 308.00 (2'424.43%)	70'932.00 (124.68%)	64'667.00 (113.67%)	1'379'
68 Median 273.50 (2'468.31%)	72'446.50 (127.34%)	65'841.00 (115.73%)	1'404'
69 Mean 354.50 (2'471.97%)	72'294.40 (127.07%)	65'780.50 (115.62%)	1'406'
70 Max 430.00 (2'524.84%)	72'739.00 (127.85%)	67'238.00 (118.19%)	1'436'
71 StdDev 36	411.21	543.26	16'610.
72 Min2opt 00 (110.47%)	62'409.00 (109.70%)	63'365.00 (111.38%)	62'846.
73 Median2opt 50 (112.55%)	63'061.00 (110.84%)	64'439.50 (113.27%)	64'032.
74 Mean2opt 06 (112.69%)	63'052.46 (110.83%)	64'535.72 (113.44%)	64'113.
75 Max2opt 00 (115.03%)	63'945.00 (112.40%)	65'715.00 (115.51%)	65'441.
76 StdDev2opt	345.01	524.40	608.33
77 MeanTime (ms)	458.28	133.08	3'374.
78 56			
79 The percentage next to the values are an indication of the performance of the heuristic.			
80 The performance is a percentage of the distance compared to the optimal distance.			
81 It should never be below 100% and the closer to 100% the better.			
82			
83 Analysis for dataset: nrw1379			
84 Optimal tour length: 56'638			
85 Metric	ClosestFirstInsert	FarthestFirstInsert	
86 RandomTour			

```

86 -----
87 Min          69'097.00 (122.00%)           62'192.00 (109.81%)           1'391'
88 Median       69'459.50 (122.64%)           62'867.50 (111.00%)           1'423'
89 Mean         69'462.54 (122.64%)           62'902.60 (111.06%)           1'424'
90 Max          69'746.00 (123.14%)           63'747.00 (112.55%)           1'463'
91 StdDev       148.74                         350.23                         17'106.
92             34
92 Min2opt      63'585.00 (112.27%)           61'517.00 (108.61%)           61'803.
93 Median2opt   63'912.50 (112.84%)           62'110.00 (109.66%)           62'832.
94 Mean2opt     63'914.38 (112.85%)           62'129.60 (109.70%)           62'834.
95 Max2opt      64'214.00 (113.38%)           63'171.00 (111.53%)           63'610.
96 StdDev2opt   148.23                         316.09                         417.34
97 MeanTime (ms) 791.38                         214.16                         6'901.
98
99 The percentage next to the values are an indication of the performance of the heuristic.
100 The performance is a percentage of the distance compared to the optimal distance.
101 It should never be below 100% and the closer to 100% the better.
102
103 Analysis for dataset: u1817
104 Optimal tour length: 57'201
105 Metric          ClosestFirstInsert          FarthestFirstInsert
106 RandomTour
106 -----
107 Min          70'227.00 (122.77%)           67'682.00 (118.32%)           2'073'
108 Median       70'938.50 (124.02%)           68'921.00 (120.49%)           2'120'
109 Mean         70'873.08 (123.90%)           68'922.26 (120.49%)           2'122'
110 Max          71'358.00 (124.75%)           70'479.00 (123.21%)           2'200'
111 StdDev       255.05                         616.04                         25'193.
111             49
112 Min2opt      63'748.00 (111.45%)           66'912.00 (116.98%)           64'636.
113 Median2opt   64'017.50 (111.92%)           67'702.50 (118.36%)           65'966.
114 Mean2opt     64'044.92 (111.96%)           67'693.66 (118.34%)           65'981.
115 Max2opt      64'409.00 (112.60%)           68'897.00 (120.45%)           67'866.
116 StdDev2opt   144.49                         487.37                         632.62
117 MeanTime (ms) 1'975.28                         636.72                         26'757.
117             16
118
119 The percentage next to the values are an indication of the performance of the heuristic.
120 The performance is a percentage of the distance compared to the optimal distance.
121 It should never be below 100% and the closer to 100% the better.

```